Wide & Deep Learning for Recommender Systems 论文阅读总结

@[toc]

Abstract

Generalized linear models with nonlinear feature transformations are widely used for large-scale regression and classification problems with sparse inputs. Memorization of feature interactions through a wide set of cross-product feature transformations are effective and interpretable, while generalization requires more feature engineering effort. With less feature engineering, deep neural networks can generalize better to unseen feature combinations through low-dimensional dense embeddings learned for the sparse features. However, deep neural networks with embeddings can over-generalize and recommend less relevant items when the user-item interactions are sparse and high-rank. In this paper, we present Wide & Deep learning, jointly trained wide linear models and deep neural networks to combine the benefits of memorization and generalization for recommender systems. We productionized and evaluated the system on Google Play, a commercial mobile app store with over one billion active users and over one million apps. Online experiment results show that Wide & Deep significantly increased app acquisitions compared with wide-only and deep-only models. We have also open-sourced our implementation in TensorFlow.

1. Introduction

1.1 Memorization & Generalization

- **Memorization** can be loosely defined as learning the frequent co-occurrence of items or features and exploiting the correlation available in the historical data.
- **Generalization**, on the other hand, is based on transitivity of correlation and explores new feature combinations that have never or rarely occurred in the past.

这个是从人类的认知学习过程中演化来的。人类的大脑很复杂，它可以记忆(memorize)下每天发生的事情（麻雀可以飞，鸽子可以飞）然后泛化(generalize)这些知识到之前没有看到过的东西（有翅膀的动物都能飞）。 但是泛化的规则有时候不是特别的准，有时候会出错（有翅膀的动物都能飞吗）。那怎么办那，没关系，记忆(memorization)可以修正泛化的规则(generalized rules)，叫做特例（企鹅有翅膀，但是不能飞）。

这就是Memorization和Generalization的来由或者说含义。

> Wide&Deep Mode就是希望计算机可以像人脑一样，可以同时发挥memorization和generalization的作用。
> —Heng-Tze Cheng(Wide&Deep作者)

现在推荐系统的一个难点就是同时实现Memorization以及Generalization，这个难点与搜索排名问题相似。

- Memorization: 之前大规模稀疏输入的处理是：通过线性模型 + 特征交叉。通过特征交叉能够带来很好的效果并且可解释性强。但是Generalization（泛化能力）需要更多的人工特征工程。
- Generalization: 相比之下，DNN几乎不需要特征工程。通过对低纬度的dense embedding进行组合可以学习到更深层次的隐藏特征。但是，缺点是有点over-generalize（过度泛化）。推荐系统中表现为：会给用户推荐不是那么相关的物品，尤其是user-item矩阵比较稀疏并且是high-rank（高秩矩阵）

两者区别： Memorization趋向于更加保守，推荐用户之前有过行为的items。相比之下，generalization更加趋向于提高推荐系统的多样性（diversity）。

1.2 Wide & Deep

本文中，介绍了一种新的方法，**Wide&Deep**，包括两部分，**Wide Part**和**Deep Part**。

- **Wide部分：利用了广义线性模型，提高可解释性。** 在大规模的在线推荐系统中，logistic regression应用非常广泛，因为其简单、易扩展、可解释性。LR的输入多半是二值化后的one-hot稀疏特征。Memorization可通过在稀疏特征上做特征交叉来实现，例如：user_installed_app=netflix，impression_app=pandora，当user_installed_app与impression_app的取值都为1时，其组合特征AND(user_installed_app=netflix, impression_app=pandora)的值则为1，否则为0。 缺点：无法学习高阶组合特征，并且需要进行人工特征工程。
- **Deep部分：主要是发现训练集中未出现的高阶组合特征。** Embedding-based模型可以在很少的特征工程情况下，通过学习一个低维的embedding vector来学习训练集中从未见过的组合特征。例如，FM与DNN。不需要进行复杂的特征工程。 缺点：当query-item矩阵是稀疏并且是high-rank的时候（比如user有特殊的爱好，或item比较小众），很难非常效率的学习出低维度的表示。这种情况下，大部分的query-item都没有什么关系。但是dense embedding会导致几乎所有的query-item预测值都是非0的，这就导致了推荐过度泛化，会推荐一些不那么相关的物品。

左侧是Wide-only，右侧是Deep-only，中间是Wide & Deep：

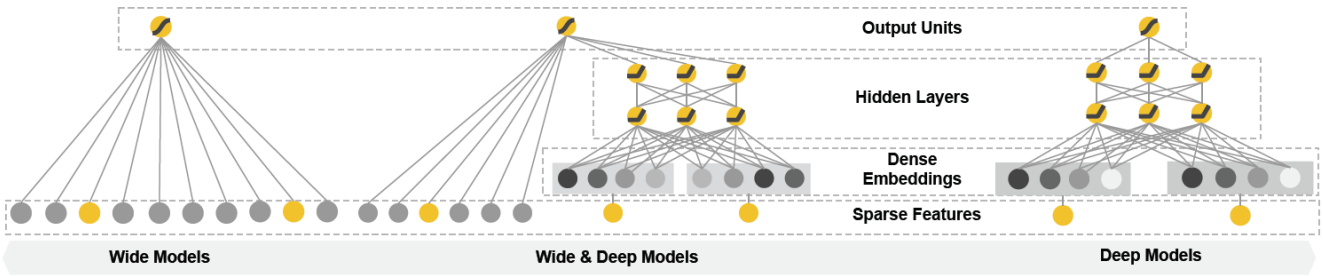


Figure 1: The spectrum of Wide & Deep models.

Wide&Deep结合以上两者的优点，平衡**Memorization**和**Generalization**。相比于**wide-only**和**deep-only**的模型，**Wide&Deep**提升显著。

2. 推荐系统

总的来说， 推荐系统 = Retrieval + Ranking

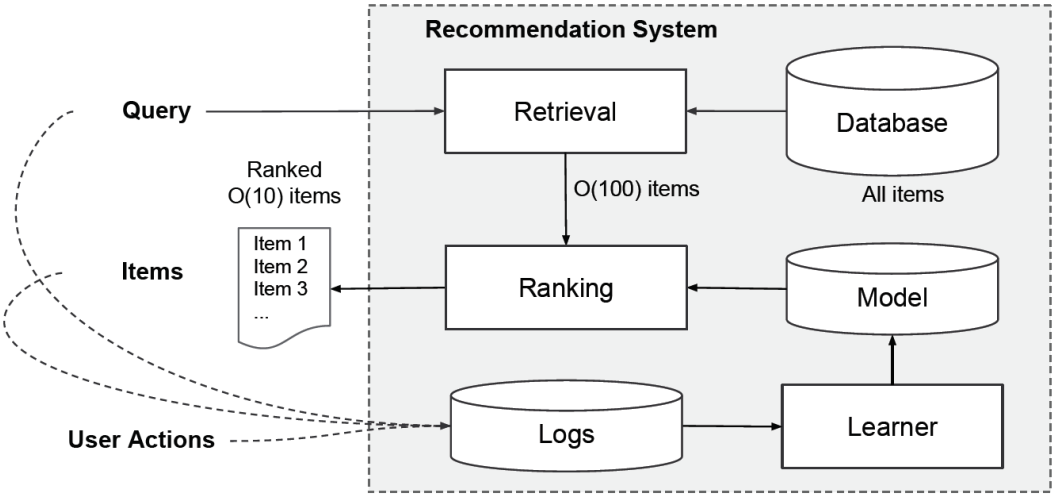


Figure 2: Overview of the recommender system.

先来看一下推荐系统的整体架构，由两个部分组成，**检索系统(或者说候选生成系统）** 和 **排序系统(排序网络)**。首先，用 **检索(retrieval)** 的方法对大数据集进行初步筛选，返回最匹配 query 的一部分物品列表，这里的检索通常会结合采用 **机器学习模型(machine-learned models)** 和 **人工定义规则(human-defined rules)** 两种方法。从大规模样本中召回最佳候选集之后，再使用 **排序系统** 对每个物品进行算分、排序，分数 P(y|x)，y 是用户采取的行动(比如说下载行为)，x 是特征，包括

- **User features** e.g., country, language, demographics
- **Contextual features** e.g., device, hour of the day, day of the week
- **Impression features** e.g., app age, historical statistics of an app

Wide-Deep Learning 就是用在排序系统中。

> "The recommender system returns a list of apps (also referred to as impressions)"，即impression即推线系统返回的推荐结果。

3. Wide & Deep Learning

简单来说，人脑就是一个不断记忆（memorization）并且归纳（generalization）的过程，而这篇论文的思想，就是将宽线性模型（Wide Model，用于记忆，Figure 1图左侧）和深度神经网络模型（Deep Model，用于归纳，Figure 1图右侧）结合，汲取各自优势形成了 Wide & Deep 模型用于推荐排序（Figure 1图中间）。

3.1 The Wide Component

Wide Part其实是一个广义的线性模型

使用特征包括：

- raw input 原始特征
- cross-product transformation 组合特征

Wide component 是一个广义线性模型形式：$y = w^T x + b$，其中y 是预测，$x = [x_1, x_2, \cdots, x_d]$ 是一个d维特征矢量，$w = [w_1, w_2, \cdots, w_d]$是模型参数，$b$ 是偏差。

其中输入特征包括：

- raw input features 原始特征
- transformed features 组合特征

其中最重要的转换是 **cross-product transformation**，定义为：

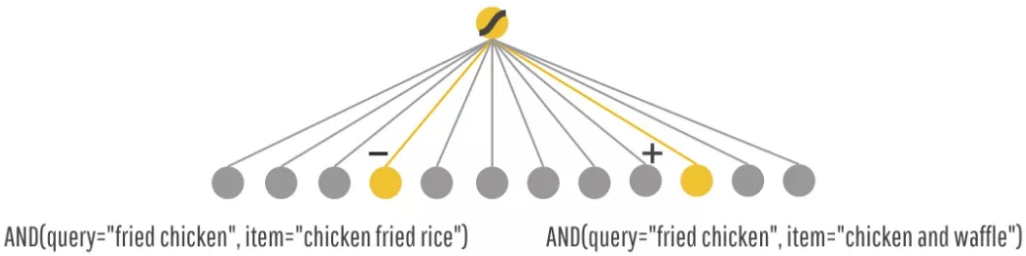
$$\phi_k(x) = \prod_{i=1}^{d} x_i^{c_{ki}} \quad c_{ki} \in \{0, 1\}$$

k表示第k个组合特征。i 表示输入X的第i维特征。C_{ki} 表示这个第i维度特征是否要参与第k个组合特征的构造。d表示输入X的维度。那么到底有哪些维度特征要参与构造组合特征那？这个是你之前自己定好的，在公式中没有体现。

饶了一大圈，整这么一个复杂的公式，其实就是我们之前一直在说的one-hot之后的组合特征：仅仅在输入样本X中的特征gender=female和特征language=en同时为1，新的组合特征`AND(gender=female, language=en)`才为1。所以只要把两个特征的值相乘就可以了。

> Cross-product transformation 可以在二值特征中学习到组合特征，并且为模型增加非线性

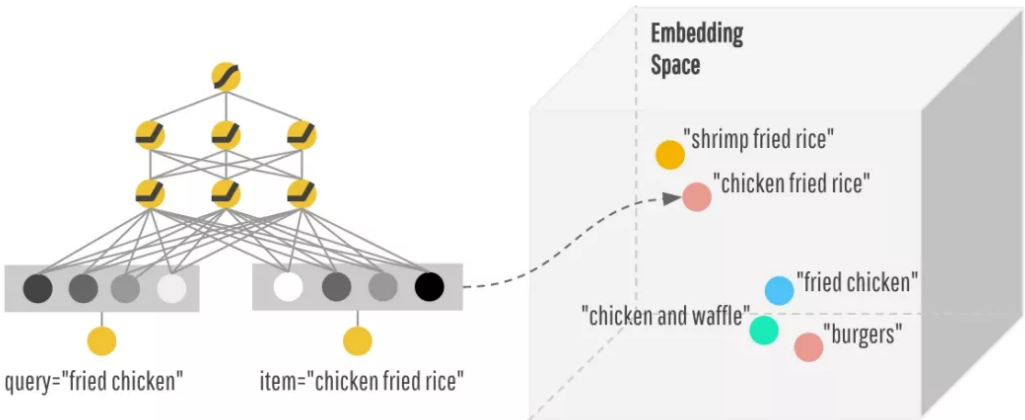
接下来我们用同一个例子来说明：你给model一个query（你想吃的美食），model返回给你一个美食，然后你购买/消费了这个推荐。 也就是说，推荐系统其实要学习的是这样一个条件概率： **P(consumption | query, item)**



Wide Part可以对一些特例进行memorization。比如AND(query=”fried chicken”, item=”chicken fried rice”)虽然从字符角度来看很接近，但是实际上完全不同的东西，那么Wide就可以记住这个组合是不好的，是一个特例，下次当你再点炸鸡的时候，就不会推荐给你鸡肉炒米饭了。

3.2 The Deep Component

Deep part 就是一个前馈神经网络



Deep Part通过学习一个低纬度的dense representation（也叫做embedding vector）对于每一个query和item，来泛化给你推荐一些字符上看起来不那么相关，但是你可能也是需要的。比如说：你想要炸鸡，Embedding Space中，炸鸡和汉堡很接近，所以也会给你推荐汉堡。

Embedding vectors被随机初始化，并根据最终的loss来反向训练更新。这些低维度的dense embedding vectors被作为第一个隐藏层的输入。隐藏层的激活函数通常使用ReLU。

一开始嵌入向量(embedding vectors)被随机初始化，然后训练过程中通过最小化损失函数来优化模型。每一个隐层(hidden-layer)做这样的计算：

$$a^{(l+1)} = f(W^{(l)}a^{(l)} + b^{(l)})$$

其中 l 是层数，f 是激活函数，$a^{(l)}, b^{(l)}$ 和 $W^{(l)}$ 分别是激活函数、偏差和第 l 层的模型权重。

基于 embedding 的深度模型的输入是

- 类别特征**(产生embedding)**
- 连续特征。

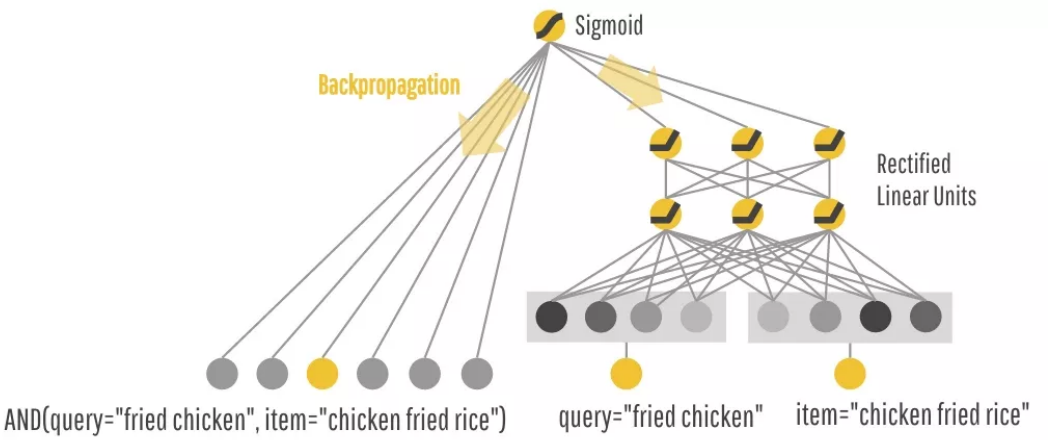
3.3 Joint Training

对两个模型的输出算 log odds ratio 然后加权求和，作为预测。

Joint Training vs Ensemble

- Joint Training 同时训练 wide & deep 模型，优化的参数包括两个模型各自的参数以及 weights of sum；
- Ensemble 中的模型是分别独立训练的，互不干扰，只有在预测时才会联系在一起。

原始的稀疏特征，在两个组件中都会用到，比如`query="fried chicken" item="chicken fried rice"`:



在训练的时候，根据最终的loss计算出gradient，反向传播到Wide和Deep两部分中，分别训练自己的参数。也就是说，**两个模块是一起训练的**，注意这不是模型融合。

- Wide部分中的组合特征可以**记住**那些稀疏的，特定的rules
- Deep部分通过Embedding来**泛化**推荐一些相似的items

Wide模块通过组合特征可以很效率的学习一些特定的组合，但是这也导致了他并不能学习到训练集中没有出现的组合特征。所幸，Deep模块弥补了这个缺点。 另外，因为是一起训练的，wide和deep的size都减小了。wide组件只需要填补deep组件的不足就行了，所以需要比较少的cross-product feature transformations，而不是full-size wide Model。

对于combined model，对于logistic regression问题，模型输出为：

$$P(Y=1|x)=\sigma(w_{wide}^T[x,\phi(x)]+w_{deep}^T a^{(l_f)}+b)$$

其中 Y 是二分类标签，$\sigma(\cdot)$是sigmoid函数，$\phi(x)$ 是原始特征x 的cross product transformation，b 是偏差；w_{wide}是所有wide model权重，w_{deep}应用到最终激活函数$a^{(l_f)}$的权重。

论文中的实现：

- 训练方法是用mini-batch stochastic optimization。
- Wide组件是用FTRL（Follow-the-regularized-leader） + L1正则化学习。
- Deep组件是用AdaGrad来学习。

4. 系统实现

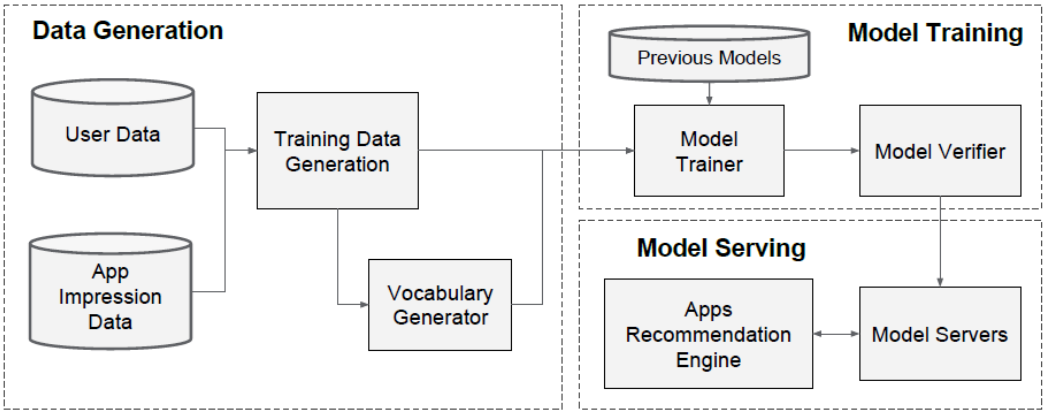


Figure 3: Apps recommendation pipeline overview.

4.1 训练数据生成

一定格外的关注训练数据到底是什么？这对于理解推荐系统到底是怎么回事很重要。 先给出结论：

一次展示中的一个**Item**就是一条样本。

样本的label要根据实际的业务需求来定，比如APP Store中想要提高APP的下载率，那么就以这次展示的这个Item中用户有没有下载，作为label。下载了label为1，否则为0. 说白了，模型需要预测，在当前Query的条件下，对于这个Item，用户下载的条件概率。

离散特征map成id

对应Figure3中的"Vocabulary Generator"，过滤掉出现次数少于设定阈值的离散特征取值，然后把这些全部map成一个ID。离散特征取值少，就直接编号。多的话可能要Hash

连续特征通过分位数规范化到[0,1]

先把所有的值分成n份，那么属于第i部分的值规范化之后的值为 (i - 1)/(n - 1)。

4.2 模型训练

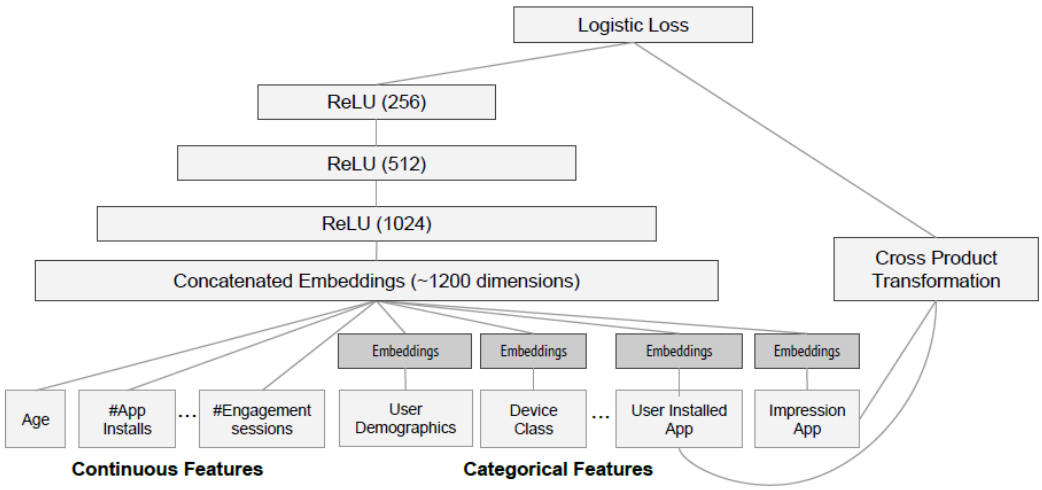


Figure 4: Wide & Deep model structure for apps recommendation.

Deep部分使用的特征：

- 连续特征
- Embedding后的离散特征，Item特征

Wide部分使用的特征：

- Cross Product Transformation生成的组合特征

> 官方给出的示例代码中，Wide部分还使用了离散特征（没有one-hot）。也有大佬说不用特征交叉效果也很好，这个大家在实际项目中就以实验为准吧。

每当有新的数据到达的时候，就要重新训练。如果每次都从头开始会非常耗时，Google给出的解决办法是：实现了warm-starting system, 它可以用之前模型的embeddings 和 线性模型的weights来初始化新的模型。

Embedding维度大小的建议： Wide&Deep的作者指出，从经验上来讲Embedding层的维度大小可以用$k\sqrt[4]{n}$来确定：n是原始维度上特征不同取值的个数；k是一个常数，通常小于10.

4.3 线上使用

模型被部署之后。每一次请求，服务器会收到一系列的app候选集（从app retrival system输出的）以及user features（用于为每一个app打分）。然后，模型会把APP按照score排序，并展示给user，按照这个顺序展示。score就是对于wide & deep模型的一次 forward pass。为了控制每一次request响应时间在10ms内，引入了并行化技术。将app候选集分成多个小的batches，并行化预测score。

5. 实验结果

从APP下载率和服务性能两个方面对wide & deep推荐系统进行评估。

5.1 APP下载率

进行了为期三周的A/B在线测试，抽取1%用户使用之前的wide模型(logistic regression + 很多cross-product feature transformations)，1%用户使用新提出的wide & deep模型，1%用户只使用deep 模型(FNN + embeding特征和连续特征)，其中实验结果如下：

Table 1: Offline & online metrics of different models. Online Acquisition Gain is relative to the control.

Model	Offline AUC	Online Acquisition Gain
Wide (control)	0.726	0%
Deep	0.722	+2.9%
Wide & Deep	0.728	+3.9%

wide & deep模型在离线测试中提升了2个百分点，而在在线测试的下载率中相比于base model，下载率提升了3.9%。

此外，实验结果发现**线上的提升比线下还多**。可能的原因是：

线下数据集，lablel 和 impression都是固定的。而线上通过组合memorization和generalization，探索更多的新的推荐；而且线上也可以通过user的反应来学习。

> DeepFM论文中也提到了，线下验证发现增长比较少，还是要看线上的结果，线上可能会比较多。

5.2 服务表现

具有高吞吐量和低延迟的服务是挑战与高水平的跟踪所面临的我们的商业移动应用商店。

考虑到高吞吐量和低延迟的服务的要求，已经移动APP商店的峰值使用量（每秒给超过1000万APP进行打分），使用了多线程并同时将将batch切分更小，实验结果如下：

Table 2: Serving latency vs. batch size and threads.

Batch size	Number of Threads	Serving Latency (ms)
200	1	31
100	2	17
50	4	14

实验结果表明，使用多线程可有效缓解服务延迟。

6. 结论

- Memorization 和 generalization对推荐系统都很重要。
- Wide linear model可以高效的memorize 稀疏特征的交互特征。
- DNN可以学习到之前没有出现过的组合特征，通过低维度的嵌入。
- Wide & Deep综合这两者，经过Google的测试，效果确实有提升。

7. 后记

7.1 Wide & Deep Model优缺点

缺点：Wide部分还是需要人为的特征工程。 优点：实现了对memorization和generalization的统一建模。

7.2 适用范围

Wide & Deep Model适用于输入非常稀疏的大规模分类或回归问题。比如推荐系统、search、ranking问题。 输入稀疏通常是由离散特征有非常非常多个可能的取值造成的，one-hot之后维度非常大。

参考文献

[1][Wide & Deep Learning for Recommender Systems](https://dl.acm.org/citation.cfm?id=2988454)

[2][简单易学的深度学习算法——Wide & Deep Learning](https://blog.csdn.net/google19890102/article/details/78171283)

[3][Wide & Deep Learning Official Example](https://github.com/tensorflow/models/tree/master/official/wide_deep)

[4][Wide&Deep理论与实践](https://blog.csdn.net/u010352603/article/details/80590129)

[5] 论文笔记 - Wide and Deep Learning for Recommender Systems

[6] 深度学习在 CTR 中应用